CS229 Project - Best Buy Recommendation System

Nikhil Rajendra, Anubhav Dewan, Mehmet Can Colakoglu

Introduction

Recommender systems have become an area of active research. They are especially important in the e-commerce industry because they help increase revenues and improve customer experience. A lot of different techniques have been explored for different kind of recommender systems based on the desired objective and the data that is available to base the recommendations on. In this project we work on developing a simple recommender system based on search query and click data. In particular, we are focusing on recommending Xbox games based on a given query. This project originated as a Kaggle[1] competition and is closed right now. We have implemented different techniques for building recommendation systems to this problem and compare how they perform and observe their strengths and weaknesses.

Data

The dataset available through the competition website consists of the query search and click data. The data is from the BestBuy mobile website consisting of clicks in the category of Xbox games. This data contains ~42,000 different clicks. The attributes provided as part of the training click data were user id, search query, product (SKU) id clicked from the query (along with additional attributes of click and query times). They also provided an XML file with different attributes for each Xbox game available with BestBuy. This data was difficult to process because it required parsing this XML file and had large amount of natural text and large numbers of overlapping and redundant product features. We chose to focus on getting insights based on the search and click data since it allowed us to explore different methods to develop recommender systems and compare their performance.

Preprocessing

String Manipulation

The most important feature of the data is the queries since they are the only user inputs provided. Being user inputs, queries exhibit noise which is generated by spelling mistakes, repeated words and excess/redundant characters. To get rid of this noise, we implemented detailed pre-processing steps.

1) All non-alphanumeric characters (except spaces) were removed
2) All queries were reduced to lower case
3) Each query was sorted with respect to the words
4) Repeated words were removed
5) Spaces were removed
6) All generic queries with the phrases “xbox”, “360” and “xbox 360” were gathered into a single, standard query

Following the above steps for each of the ~42,000 queries, we ended up with ~3,800 unique queries. The clicks and corresponding product data was aggregated accordingly. This preprocessing is also done on the test set queries.

Before: “Batman: Arkham City”
After: “arkhambatmancity”

[1] http://www.kaggle.com/c/acm-sf-chapter-hackathon-small

Spell Checking with Clustering

Although the results from the pre-processing method detailed above resulted in significant reduction of the number of queries, we explored if we could improve even further. We implemented a clustering method to group queries. Specifically, we used K-means clustering with the Levenshtein[2] distance as the similarity metric.

This approach was implemented successfully and was particularly useful for grouping together longer queries which are close to each other. The closeness of two reasonably long queries is most likely a result of spelling mistakes and with this method, we were able to group these queries together and reduce the number of unique queries from ~3,800 to ~500. However there were a few issues:

1) The choice of number of clusters k is not straightforward and we relied on trial-and-error in choosing k. This might be alleviated by using other, hierarchical clustering methods but they also require a threshold for termination which is again not straightforward to choose.
2) For longer queries, this method works very well however for short queries, results aren't as good. The reason for this is that the difference between short queries such as "rise" and "riot" are small enough for them to be clustered together by the algorithm.

Due to these issues, we decided not to include this spell checking method into our recommendation system at this time.

Sample cluster:

batman arkham city	batman arkham city	batmsn arkham city	batman arkhan city
batman arhkam city	batman arkhham city	batman arkhum city	batman arkham cita
batman: arkhum city	batman arkaham city	batmart: arkham city	batman arkham cuty
batman arkhamn city	batman asrkham city	batman archam city	batman arkyam city
batman arkman city	batman arcyum city	batman arkhym city	baseman arkham city
batman arlham city	batman arkhem city	batman adkham city	batman arckam city
batmam arkham city	batman arkuam city	batman arklym city	batman arkahm city
batman arkam island	batman arklum city	batman arkam city 2	batman arkham cith

Methodology

Notation

We represent the query – item relationship with the following notation:

$QI: Query - Item\ matrix$

$QI_i = [c_{i1}\ c_{i2} \dots c_{ij} \dots\ c_{im}]$

$QI_{ij} = c_{ij}: \#\ of\ times\ SKU_j\ was\ clicked\ after\ Query_i$

Collaborative Filtering

For recommender engines collaborative filtering is used to identify different users which are similar to a particular user. Then these similar users are used to recommend products that were popular with them. In the problem we work on, the notion of users is very weak because we have no additional information about the users except for the products they clicked on. Also the number of clicks per user is very low (we have ~38,000 users and 42,000 queries in the training data). Hence we decided to use collaborative filtering replacing the notion of user by search queries and try to identify similar queries to each query in the training set. We also use the notion of finding similar items to what was searched for using the new query and recommend popular items among similar items (this is sometimes called as item to item collaborative filtering). These two approaches are highlighted below:

[2] Levenshtein distance between two strings is the minimum number of single character edits required to transform one to the other

Query based Collaborative Filtering

This is used to make recommendations based on similar queries used before in the training set. For finding similar queries based on products clicked on, we use the query item matrix. To measure the similarity between different queries, we used cosine similarity between the corresponding rows of QI.

$$similarity\left(\overrightarrow{QI_i}, \overrightarrow{QI_j}\right) = cos\left(\overrightarrow{QI_i}, \overrightarrow{QI_j}\right) = \frac{\overrightarrow{QI_i} * \overrightarrow{QI_j}}{\|\overrightarrow{QI_i}\| \times \|\overrightarrow{QI_j}\|}$$

This has one problem since the similarity between different queries is heavily influenced by very popular items and the less popular items become less relevant. To counter this we used the inverse frequency approach where each column of QI was scaled by the total number of clicks for that item in the training set. Using this, a query to query similarity matrix was constructed for the training set queries.

$$QQ: Query - Query\ similarity\ matrix; \quad QQ_{ij} = similarity\left(\overrightarrow{QI_i}, \overrightarrow{QI_j}\right)$$

Queries come from an infinite space and given a new query in the test set, it may not map to a query in the training set. Preprocessing of queries was useful to reduce the number of unique queries we had in the training set. Given a new query in the test set, we find the closest matching query using normalized Levenshtein distance.

Based on collaborative filtering, we should use similar queries and recommend top 5 items corresponding to these similar queries that were clicked on the most. To achieve this we calculate the weighted sum of all QI_j using QQ_{ij} as the weights.

$$Q_{sum} = \sum_{j=1}^{n} \left(QQ_{ij} * QI_{j.}\right)$$

After calculating this vector we recommend the top 5 SKUs with the highest combined score.

Item to Item Collaborative Filtering

Another way of using collaborative filtering is to identify items that are similar to the item a new query is searching for. This approach is useful in many e-commerce applications because it allows retailers to recommend products in different categories based on the different items that have been purchased. This has also been shown to have better scalability for online recommendations compared to user based collaborative filtering. In our problem we find out item to item similarity by using QI^T. Each row of this matrix corresponds to an SKU and this matrix stores the number of times the SKU was clicked based on each query. The item to item similarity matrix is then constructed.

$$II: Item - Item\ similarity\ matrix; \quad II_{ij} = similarity\left(\overrightarrow{QI_i^T}, \overrightarrow{QI_j^T}\right)$$

Unlike many recommender systems we don't have data of different products that a user bought and we need to map a new test query to an existing SKU item. We use a roundabout way to do this. Given a new query we find the closest query in the training set. We know the products that were clicked on through the query. We use the weighted average of the item similarity rows corresponding to the clicked SKUs to get a combined similarity vector corresponding to the new query.

$$I_{sum} = \sum_{j=1}^{n} \left(\frac{QI_{ij}}{\sum QI_i} * II_{j.}\right)$$

We recommend top 5 items with the highest combined similarity score.

Clustering

Cluster models are also widely used in recommender systems. Normally, a recommendation engine would cluster users based on their browsing or purchasing information, and recommend items to a user based on other users in the same cluster. In our case, the user-related information is too sparse for that to be truly useful as mentioned above. Therefore we use query information instead of user information for the clustering implementation. Besides clustering queries, we also implemented an item clustering method. Following are the two approaches:

Query Clustering

Cosine similarity as explained above is used as the similarity metric and K-means clustering is used as the clustering algorithm. Here, the choice of k, the number of clusters, is an important one. Since we have a product catalog of ~800 items (not all of them clicked), the number of clusters is chosen to be considerably smaller than that but still large enough to reflect the diversity of the queries (k~=500). After that, running the algorithm until convergence creates the query clusters.

For a new query in the test set, we match it to the closest query in the training set. Since we have clustered the queries, this match also matches the new query to a cluster. Finally, we output the 5 top SKUs in that cluster. SKU popularity is based on the aggregated clicks for the clusters.

Sample Cluster:

```
[1] "Battfield"
[1] "battlefield bad conpany 2"
[1] "Battfield bad company 2"
[1] "Battle field bad company"
[1] "Batlefeild bad company"
[1] "Battlefield on sale"
[1] "Battlefield bad  2"
[1] "Battlefeild 2"
[1] "Battledielsd bad company 2"
[1] "bad meets evil album“
```

Item Clustering

It is also possible to use clustering methods on items in the product catalogue. QI^T constitutes an Item – Query matrix. We can use K-means clustering with cosine similarity between item rows to cluster the items. Same issues regarding the choice of k are valid in this case and are handled similarly to the query clustering (k~=80). Running the algorithm until convergence creates item clusters.

For a new query in the test set, we use string distance to match it to the closest query in the training set. Let $QI_i = [c_{i1}\ c_{i2} \dots c_{ij} \dots\ c_{im}]$ be the corresponding row of QI. For all $c_{ij} > 0$, we identify all the corresponding SKUs. We select the SKU with the maximum number of clicks and recommend the top 5 SKUs in its cluster. If we find less than 5 items in the cluster, the most popular SKUs are recommended in the remaining slot.

Sample Cluster:

```
[1] BioShock 2 Limited Edition - Xbox
360
[1] BioShock 2 - Xbox 360
[1] BioShock Infinite - Xbox 360
```

Evaluation Metric

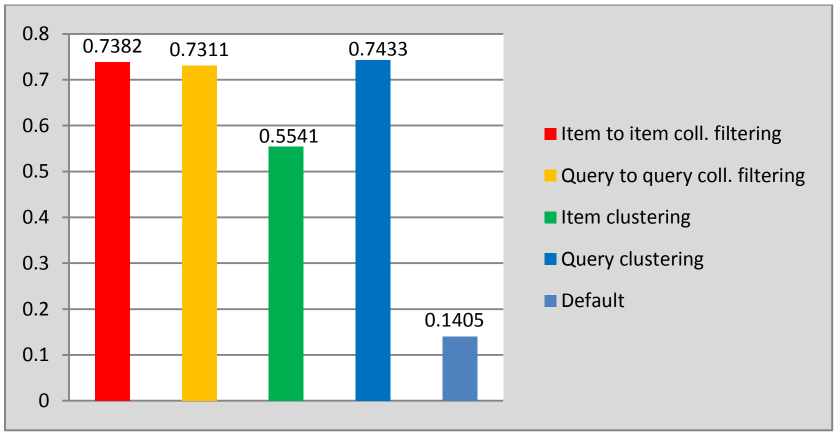
Mean Average Precision (MAP): This metric was specified by Kaggle. It is one of the standard information retrieval methods to evaluate ranked list of documents corresponding to a given query. Average precision is the average of the precision of k retrieved documents. Mean average precision is the mean of average precision over all queries. In our problem we recommend 5 products for a given query. In the test set labels, we have 1 product corresponding to each query. Thus average precision is defined as follows:

$$AP_i = \begin{cases} 1/j\ ;\ if\ the\ clicked\ item\ is\ recommended\ in\ order\ j \\ 0\ ;\ if\ item\ is\ not\ included\ in\ the\ recommendation \end{cases}$$

$$MAP = \frac{\sum_i AP_i}{size(test\ set)}$$

Results and Conclusions

We compare results of the four approaches suggested and a benchmark approach from the competition. The MAP scores are shown in the table below:



In the benchmark, 5 most popular SKUs are recommended, irrespective of the query. All our methods perform significantly better than the benchmark. Collaborative filtering for queries and items performs similarly and query based clustering performs slightly better. Item based clustering doesn't perform as well. This happens primarily since we were not able get good clustering based on items.

Future Considerations

In the data preprocessing step we explored an approach for spell checking based on clustering but were not able to integrate that into the final recommendations. Such a step would be helpful in reducing the number of unique training queries.

Given a new query we find the closest query in the training set based on modified string distance and base our models on that. For some queries, this does not work so well because we are not able to find a suitably close query (preprocessed). Hence there is scope for trying different approaches to model the query strings (based on keywords) and try to make recommendations using the given SKU attributes (for these cases).

Also we developed 3 models (Query and Item CF and Query Clustering) which perform reasonably well. There is scope of developing a composite algorithm that uses the techniques from all the methods and can provide a better performance.

References

- Linden, G., Smith, B., & York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, *7*(1), 76-80.
- Horvitz, E. (2001). Principles and applications of continual computation. *Artificial Intelligence*, *126*(1), 159-196.
- Schafer, J., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. *The adaptive web*, 291-324.
- Chen, A. Y. A., & McLeod, D. (2005). Collaborative Filtering for Information Recommendation Systems. *Encyclopedia of Data Warehousing and Mining. Idea Group*.
- Zhu, M. (2004). Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*.